Jordi Magdaleno Maltas

Màster universitari en Automàtica i Robòtica

Maig 2015

Treball de Fi de Màster

Màster universitari en Automàtica i Robòtica

Real time 3D reconstruction of non-structured domestic environment for obstacle avoidance using multiple RGB-D cameras

MEMÒRIA

Autor: Jordi Magdaleno Maltas
Director/s: Joan Aranda Lopez

Convocatòria: Maig 2015

ETSEIB

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

UPC

Abstract

This thesis presents how to obtain a real-time 3D reconstruction of a non-structured scene using multiple RGB-D cameras with the aim to use it for robot planning.

Several techniques have been developed, starting from raycasting. In order to obtain better speeds some point filtering procedures have been used.

Finally, a temporal filtering technique is used to replace the raycasting approach, and to provide a better reconstruction, a sparse points filter is implemented.


ETSEIB

Contents

ETSEIB


ETSEIB

Glossary

Alpha Parameter for the temporal filter. It determines how much probability to decrease in every voxel at the end of processing a frame. 7

CPU Central Processing Unit. 7

Epsilon Parameter for the temporal filter. It determines how much probability to increase to a voxel that has a point marking it occupied. 7

Extrinsic calibration Determine the position and orientation of the camera respect to the world reference frame. 7

GPU Graphics Processing Unit. The CPU of the video card. 7

Intrinsic calibration Calibration of the intrinsic parameters of the cameras, as the camera matrix and the lens distortion coefficients. 7

IR Infra Red. 7

NaN Not a Number. A special type to represent that a numeric operation had an invalid result. 7

Octomap A representation that uses OCtrees. It is used as the representation of the reconstruction. 7

Raycast Technique that consist in casting a ray from a point to the camera direction to locate free space in the reconstruction. 7

RGB Red, green, blue, depth. 7

RGB Red, green, blue. 7

ROS Robot Operating System. 7

ROS MoveIt! A package to develop mobile robots. 7

ROS node A ROS program. 7

ROS package A set of ROS nodes. 7

ToF Time of flight. A kind of RGB-D camera that uses the time between a light is emitted until it returns to calculate the depth. 7

Topic The channel that ROS uses for the producer-consumer messages. 7

List of Figures

1 Preface

1.1 Project origins

This work is inside a project to help disabled or elderly people to be able to be self-sufficient introducing robots in their homes.

The work explained in this thesis comes by the need to be able to provide to a set of robots the environment where they have to work to be able to plan their trajectories without colliding with other objects, and more important, with people.

1.2 Motivation

Nowadays, the expected expectancy lifetime is growing. This will lead to an increase of the elder population which in some cases can not be self-sufficient in their home.

The introduction of robotics in the home environment can be a good way to make these people self-sufficient, and at the same time help disabled people.

However, the introduction of robotics in the home environments have some problems that are different than the industrial environments and they are not addressed.

The introduction of robots to help in the kitchen is very useful, but they will operate near people, and thus, they must be able to perceive the kitchen in short time to be able to plan trajectories where they don't collide not only with objects, but also the people they are trying to help.

2 Introduction

2.1 Scope

The introduction of the robots in the home environment can help elderly and disabled people.

The usual techniques for robot interaction with the environment make necessary to have structured environments, this is to say, known environments or environments where some marks are placed so the robot can recognize them and operate in the desired way. While in the industry there aren't much problems to adapt the environment to the robot, in the home scenario the owners want the robot to adapt to the environment, thus, having all the freedom to change the environment as they please.

This makes necessary to create new perception capabilities for the home robots so they could sense these non structured environments in real time.

In this work we introduce a technique to obtain a real time 3D representation of the world using multiple ToF (time-of-flight) cameras, so that a robot can use it to help people in a safe way in a non-structured environment as a kitchen.

2.2 Objectives

It is desired to obtain:

1. A real-time 3D reconstruction using RGB-D cameras.
2. 3D reconstruction with free, occupied and unknown space modelled.
3. Be able to determine fast if a certain area is occupied.
4. Integration with robots using ROS MoveIt!

5. Be able to tune the reconstruction according to the computational resources available.

6. Be able to stream the reconstruction to make possible to distribute the work load of the main computer that controls the robot.

3 Types of RGB-D cameras

There are two kind of 3D cameras of extended use.

3.1 Structured light

Depth cameras based in structured light project a pre-defined light pattern which is used to obtain the depth distance using the distortion in the projected pattern. Normally this cameras have a RGB camera, an IR projector and an IR camera. The IR projector projects the structured light which is captured by the IR camera while the RGB camera captures the scene. The IR image is processed to obtain the depth data and is aligned with the RGB image. Sometimes, successive projections of coded or phase-shifted patterns are often required to extract a single depth frame, which leads to a lower frame rate and the need to be observing, still or slow moving objects to avoid blurring. The structured light cameras are sensitive to the ambient light, this makes them better suited for indoor environments than for outdoor.

3.1.1 Kinect

A good example of how a structured light camera work is kinect. The IR camera and the IR projector form a stereo pair with a baseline of approximately 7.5 cm. The IR projector sends out a fixed pattern of light and dark speckles. The pattern

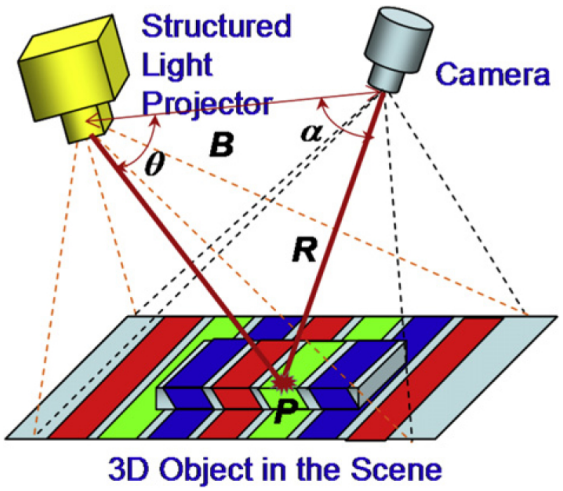

Figure 1: From [5] Diagram of how the structured light technique works.

is generated from a set of diffraction gratings, with special care to lessen the effect of zero-order propagation of a center bright dot. The depth is calculated using

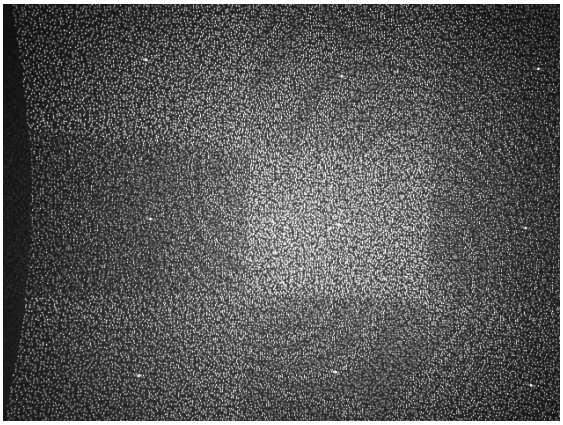

Figure 2: From [8] Kinect IR pattern.

triangulation against a memorized pattern at a known depth. For each pixel in the IR image, a small correlation window is used to compare the local pattern at

that pixel with the memorized pattern at that pixel and 64 neighbouring pixels in a horizontal window. The best match gives an offset from the known depth, in terms of pixels: this is called disparity. The Kinect device performs a further interpolation of the best match to get sub-pixel accuracy of 1/8 pixel. Given the known depth of the memorized plane, and the disparity, an estimated depth for each pixel can be calculated by triangulation.

3.2 Time-of-flight

A time-of-flight camera (ToF camera) is a range imaging camera system that resolves distance based on the known speed of light, measuring the time-of-flight of a light signal between the camera and the subject for each point of the image. The ToF camera not only receives the light projected by itself but also the ambient light, which decreases the signal to noise ratio (SNR) if used in environments with high illumination. To overcome this problem, the source light is pulsed or modulated by a continuous-wave (CW). The pulse is very used because of the ease of this technique, but in devices that has to offer great accuracy the most used is the modulation. The frequency is an important parameter. It determines the accuracy of the devices and the maximum measurable distance. High frequencies achieve better accuracy, but at the expense of the maximum measurable distance whereas low frequency has a better range but lower accuracy. The most advanced ToF cameras employ multi-frequency techniques to extend the distance without reducing the modulation frequency. This is performed adding more modulation frequencies to the mix. Each modulation frequency will have a different ambiguity distance, but the true location is the one where the different frequencies agree. This frequency is called the beat frequency.

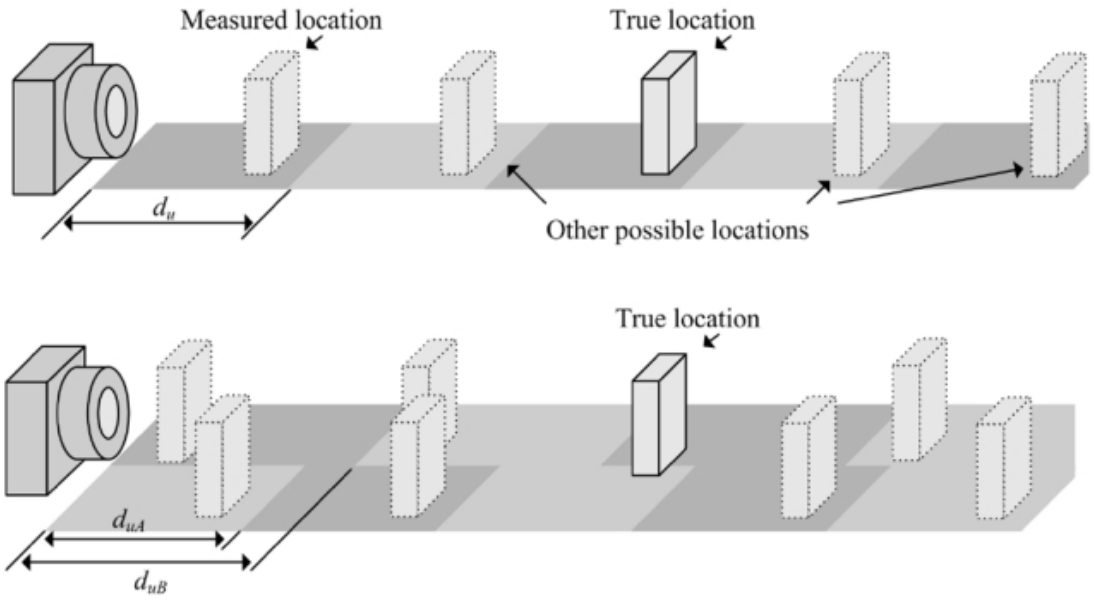

Figure 3: From [7] Operation of a multi-frequency ToF camera.

Table 1: Comparison between structured light techniques and Time-of-flight

CONSIDERATIONS	STRUCTURED-LIGHT	TIME-OF-FLIGHT
Software Complexity	Medium	Low
Material Cost	High	Medium
Compactness	High	Low
Response Time	Slow	Fast
Depth Accuracy	High	Medium
Low-Light Performance	Good	Good
Bright-Light Performance	Weak	Good
Power Consumption	Medium	Scalable
Range	Scalable	Scalable

4 Requirements

4.1 Camera requirements

As it has been explained in chapter 3, there are available two big groups of RGB-D cameras: structured light and time-of-flight.

As it is necessary to capture a big scene that is not visible only with a single camera, it is desirable that the cameras don't produce interferences between them.

Structured light cameras project and detect an IR pattern. The projection of this pattern cause some errors in the other cameras pointing at the same scene as the other cameras detect some of this pattern points as pattern points of themselves. This cause the other cameras to have some errors in the depth calculation. Even if there are some techniques to reduce the interferences as the ones presented in [12], [4] [2], time-of-flight cameras are better suited for this kind of set up because they don't interfere with each other, or the interference is very limited.

The work started using two asus xtion-pro live and their interference problems were solved with the shake and sense technique, but the release of the kinect 2 cameras and their better resolution along with little interferences between cameras lead to change the asus cameras for the kinect2. Finally, two kinect2 time-of-flight cameras are used. Nevertheless the system could be composed of more cameras and even mix the two types if it were necessary.

4.1.1 Reconstruction requirements

It is desired that the reconstruction can be used efficiently to plan trajectories for robots. This reconstruction has to be able to identify very fast if a certain area is occupied, empty, or it is unknown, that is, any camera can't see the area.

RGB-D cameras, normally give as an output a depth image from which is easy to calculate a point cloud.

Even if the point cloud can be considered a 3D Reconstruction, this representation don't accomplish with the efficiency requirements for robot planning. A widely used representation in robotics for spaces is Octomap [6]. This representation creates a tree structure where the space is discretized in voxels (cubes that represents a cube in the 3D space) where the state of the voxel is determined in a probabilistic way.

The robots that has to use the 3D reconstruction use the ROS MoveIt! framework. As it is wanted to perform an integration between the 3D reconstruction and this framework, it is interesting to use this framework to perform the 3D reconstruction.

Luckily, ROS MoveIt! provides certain plugins to be able to create a 3D representation of the world using octomap along RGB-D cameras providing depth images or point clouds. Moreover, the implementation is able to filter the robot if the 3D model of it is available.

The 3D reconstruction will be performed creating a ROS MoveIt! plugin that will use the octomap representation.

5 Environment setup

5.1 Position of the cameras

The camera positions will depend on how many cameras are available, the area of the scene that a camera can cover and the estimated amount of occlusion that the camera can suffer from the robot and the objects on the scene in the selected position.

The kitchen that has to be reconstructed can be seen in figure 4 along with a cartesian robot and a robotic arm.

The cameras are placed where they have the best coverage of the table while

Figure 4: Kitchen to be reconstructed. The camera positions are marked.

having little shadows produced by the robots and have some overlapping in their views.

For environments where a lot of cameras are available it is possible to look up [9] where there is a complete study about the set up of large multi camera systems.

5.2 Multiple Kinect2

To be able to use the kinect2 cameras, the ROS package iai_kinect2 [13] developed in the Bremen university has been used. This package, however, didn't had the possibility to use multiple cameras using a single computer, as it always used the default camera.

This would have leaded to have to use one computer per camera and perform the complete reconstruction in one of them, having some network lag and making

harder to achieve the real time goal.

To solve this problem, a contribution to the project has been done providing a code which gives to the user the possibility to select which of all the connected cameras in the computer the node has to start. This enables to run some nodes starting the different cameras connected to the computer. In this way, there isn't any network lag.

5.3 Calibration

In order to obtain a precise reconstruction, it's necessary to perform the intrinsic and extrinsic calibration of the cameras. The intrinsic calibration has been performed with the method provided by the ROS iai_kinect2 package which gives a file for the RGB and IR cameras, each of them containing the camera matrix, the distortion coefficients of the lens, the rotation matrix and the projection matrix. Also another file is generated containing the rotation and translation from the RGB camera to the IR along the essential and fundamental matrices to relate the two camera images.

For the extrinsic calibration the ROS package ar_track_alvar [10] has been used. This package provides nodes (ROS programs) to perform the tracking of some tags. The idea is to use it to provide the transformation between the camera frame to a tag which lies in a known pose in the world reference frame that can be seen by all the cameras. Once the transformation is obtained, the transformation from the world reference frame to the camera frame is calculated and stored to a file using the ROS package ros-tf-graph [11]. In this way it is obtained an extrinsic calibration that don't need to have the tag in the scene when the system is reconstructing.

However, in practice the method don't work out of the box. The tag detection has certain noise that makes that the transform is not correctly calculated and the

noise has to be eliminated. The noise is treated as white noise. A ROS node has been developed to calculate the mean of all the transformations from the camera reference frame to the tag reference frame.

The mean translation is the mean of all their components, while the rotation mean is the mean of the quaternions. This mean transform is published to the /tf topic (the rotation and translation from one reference frame to another is published in a producer-consumer style) and with the ros-tf-graph package the transform from the tag to the camera is obtained. As the pose to the tag is known in the world reference frame, it is possible to start a tf static_transform_publisher node with this transform along with a node of the ros-tf-graph that will publish the transformation from the tag position to the camera. In this way the transform from the world to the camera is possible and the extrinsic calibration is complete. It is even possible to use again the ros-tf-graph to calculate the transform from the world reference frame to the camera reference frame and don't have to use the tf static_transform_publisher node.

6 Reconstruction

The reconstruction is performed individually for every camera and thanks to the calibration, the processed frames can be applied to a common octomap. The point clouds extracted from the depth images of the cameras are used to build the reconstruction. This point clouds are in the format used by the point cloud library (PCL)[1] and can contain invalid points represented as "not a number" (NaN).

In the search of real time performance, some techniques have been implemented starting from the raycast approach given in the ROS MoveIt! framework (see section 6.1).

The steps for the reconstruction vary depending on the technique used, but

there are four steps that are always performed for all the points in the point cloud.

- Check that the points are valid (they are not NaN).
- Transform the points from the camera reference frame to the world reference frame.
- The points are checked if they are part of the robot model.
- The points that are part of the robot model are marked as free area in order to not see the robot as an obstacle (the self-collision of the robots is outside the scope of the reconstruction), while the rest are marked as occupied.

There are two main reconstruction techniques. The raycast and the temporal filter. They can use the following techniques to speed up the reconstruction:

1. Filter out points limiting the max range.
2. Point subsample.
3. Table and unreachable area subtraction.

A code optimization has been performed to the raycast implementation from which the temporal filter also takes profit.

The temporal filter, as it is less accurate than the raycast, can use a sparse point filter to obtain a better accuracy in the reconstruction at the expense of some computation time. Finally, the temporal filter can send the octomap over the network, enabling the possibility to have a dedicated perception server.

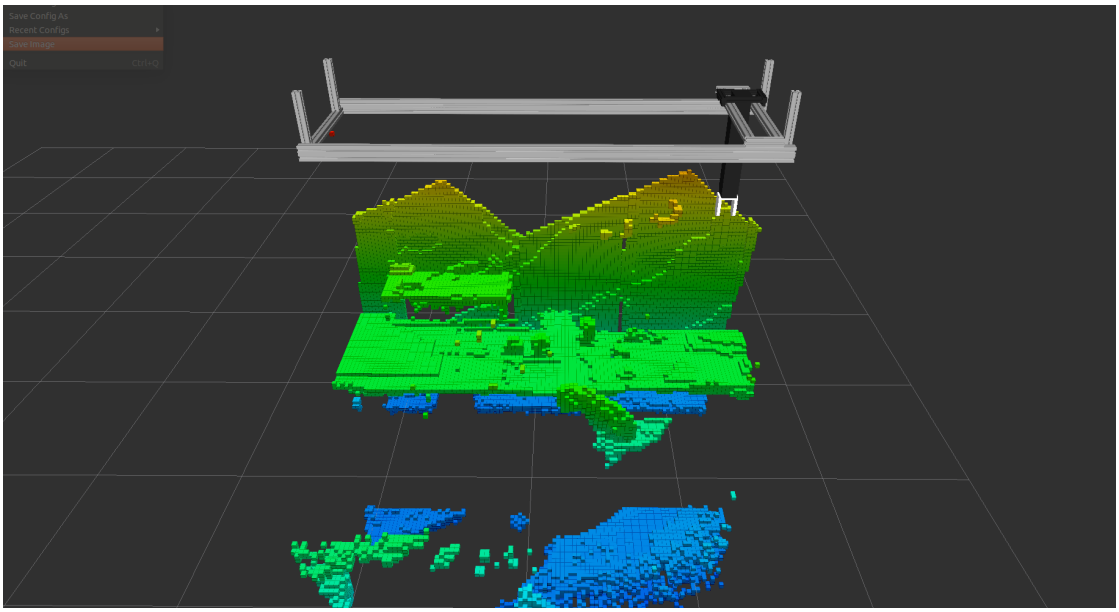

Figure 5: Reconstruction of the scene using raycast.

6.1 Reconstruction using raycast

The raycast technique consist in projecting a ray from a point to a direction until it "collides" or it arrives to a certain distance.

The idea of the reconstruction using raycast is that if a given point is seen by the camera, the line between the point and the camera origin is free. In this case, the point is an occupied space of the scene, while the ray casted from the point to the camera traverses free space.

In this approximation the points in the point cloud will increase the probability that the voxels where they lie are occupied, while the voxels along the ray increase the probability that the voxels are free.

Despite the fact that this technique models the scene in an accurate way, it's very expensive computationally. In the case that the octomap resolution is high (the voxels are small), the raycast approach is not able to provide real time recon-

struction if the quantity of points in the point cloud is high.

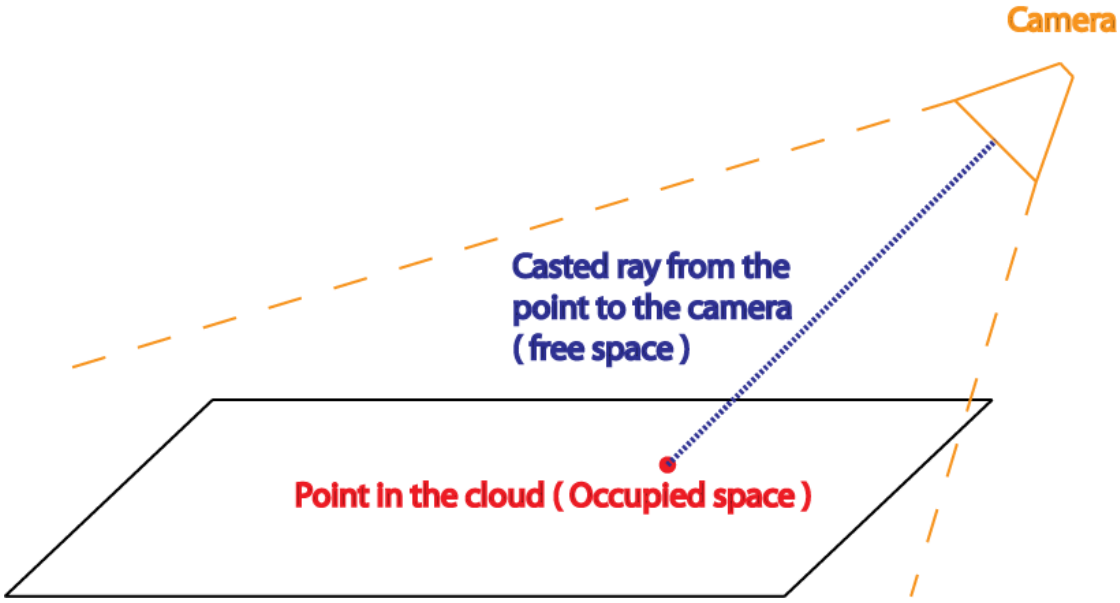

Figure 6: Explanation of the raycast technique.

6.2 Filter out points limiting the max range

In order to try to reduce the computational power necessary to reconstruct the scene using raycasting it is possible to filter out points based on the distance at which they lie. Points further than a maximum distance threshold will be filtered out. This parameter is set per camera and will alleviate the computation needs as fewer rays have to be computed.

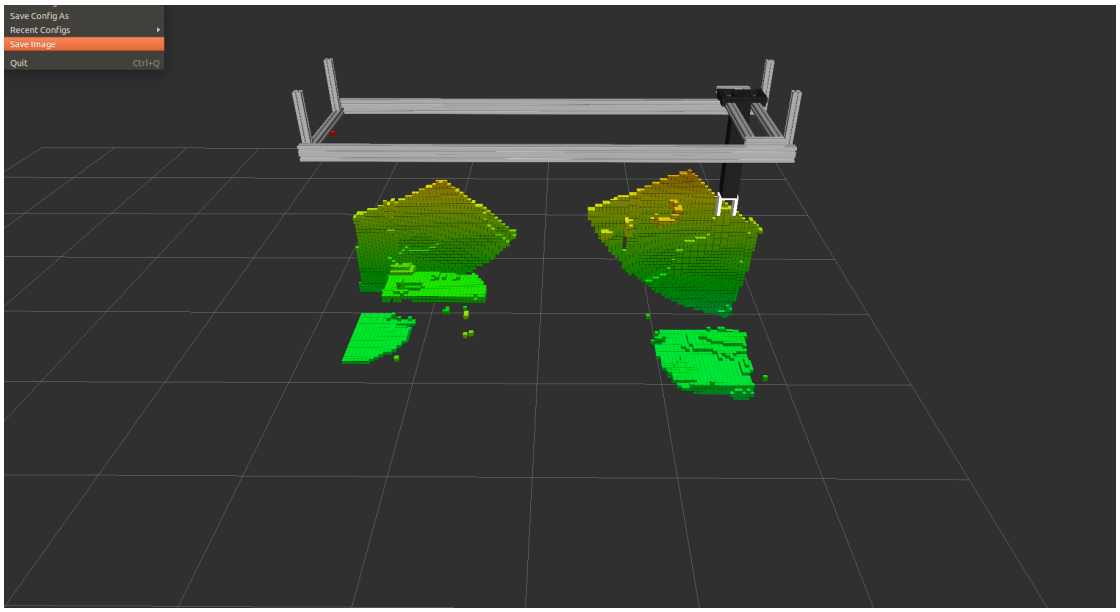

Figure 7: Reconstruction with max range limited to 1m in both cameras.

6.3 Point subsample

Depending on the resolution of the octomap and the distance from the camera to the scene, it is possible that some points in the point cloud lie in the same voxel. This permits that a camera don't need to process all the points to perform a correct reconstruction using the octomap.

Taking advantage of this, it is possible to define a parameter to skip some points of the point cloud. Being a parameter for a camera it is possible to have a high subsample in cameras where the skip of the points is not affecting very much the reconstruction accuracy (alleviating the computational cost of the processing of the point cloud) and have the rest processing all the points.

As the point cloud received from the cameras is ordered in an array as the depth image, the implemented subsample skips N-1 number of columns an N-1 number of rows. A subsample of $N = 1$ will process all the points.

As can be seen in figures 9, 20, and 11, a high value can produce holes to the reconstruction, hence the value has to be selected carefully.

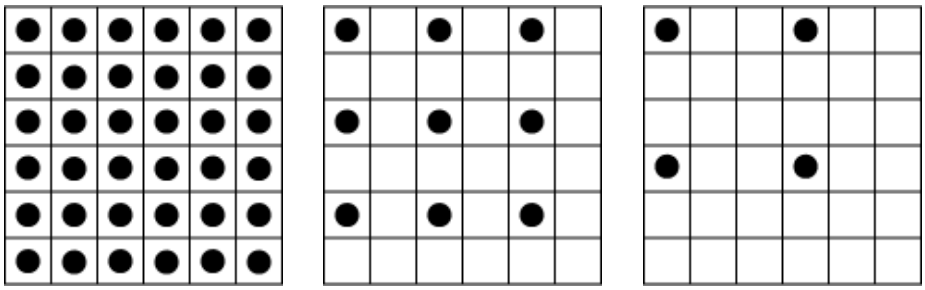

Figure 8: Image points selected with different point subsamples. From left to right, point subsample 1, 2, 3.

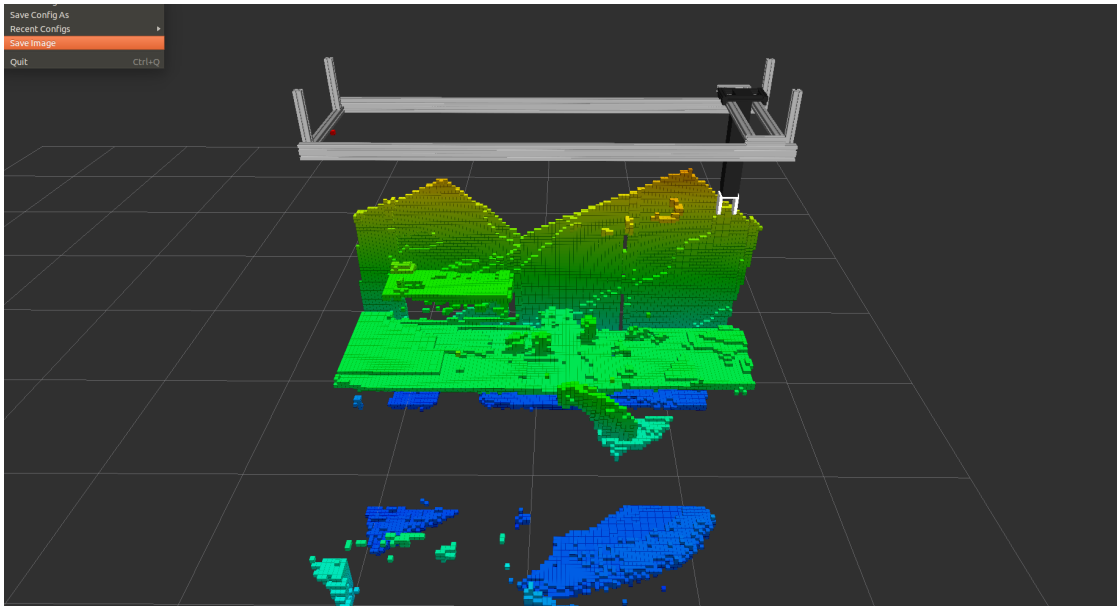

Figure 9: Scene reconstruction using point subsample = 1.

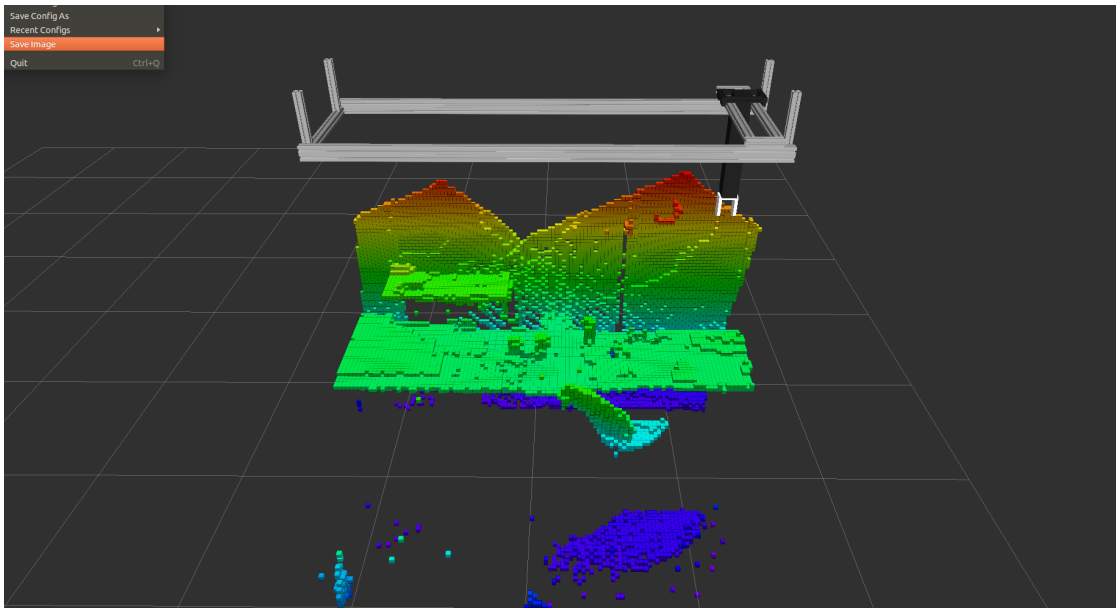

Figure 10: Scene reconstruction using point subsample = 4.

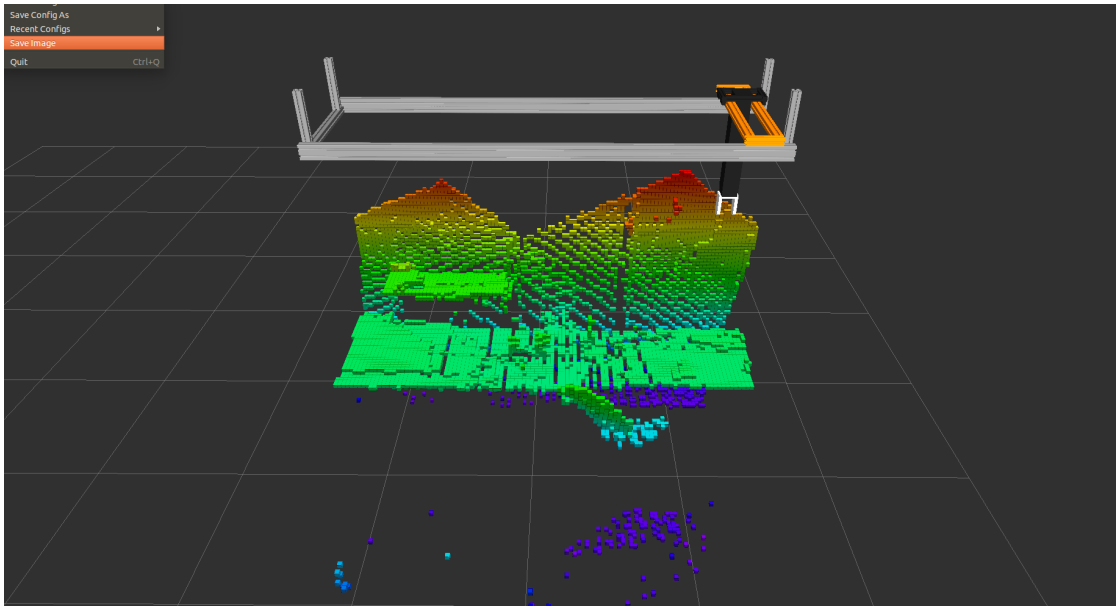

Figure 11: Scene reconstruction using point subsample = 8.

6.4 Code optimization

The implementation code provided by the ROS MoveIt! framework is analysed and some optimizations are found to be applicable.

6.4.1 Code motion of loop invariant instructions

There is a loop that consist of a main loop and an inner loop used to traverse the point cloud. In the main loop there is an iterator that is always constructed with the same value. It is possible to move away the construction of the iterator from the main loop and changing a little bit the code make it continue to traverse the point cloud in the same order.

6.4.2 Loop fusion

After putting away the constructor from the loop, the inner loop can be fusioned to the main loop maintaining the point cloud traverse order.

6.4.3 ROS queue

In ROS, by default, the messages coming from the subscribed topics go to a general queue. This queue is processed in a FIFO schedule. This means that if a new camera message arrives, it has to wait until the previous messages are treated.

As it is desired to have top priority on the camera messages, for each camera topic a dedicated queue is used to process the camera messages as soon as possible in threads. The queue length selected is set to one to have always the most recent information.

6.5 Table and unreachable area subtraction

In order to try to reduce even more the computational power necessary to reconstruct the scene using raycasting, an attempt to reduce the number of points is performed taking out of the cloud the points that lie in the areas where the robot can't reach.

It is also possible to take out the points of the table, which is assumed to be always in the same position.

Even if at first could be seen as something that would lead to an incomplete reconstruction as the filtered points will not be used in raycasting and thus the free space would be incorrectly modelled as unknown space, this is not a problem in the Ros MoveIt! configuration used by the robots, as the robot planners are configured to treat unknown space as free space. This fact provokes that in order to use this technique, there has to be enough cameras to observe the scene to avoid collisions of the robot in unknown areas.

To be able to remove the mentioned points, a ROS MoveIt! plugin has been developed. This plugin determine the plane in which the table lies and create a collision object that will filter the table and the points that are under it and at the same time will provide to the robot a surface to place the kitchen objects as the milk, sugar, frying pans, etc.

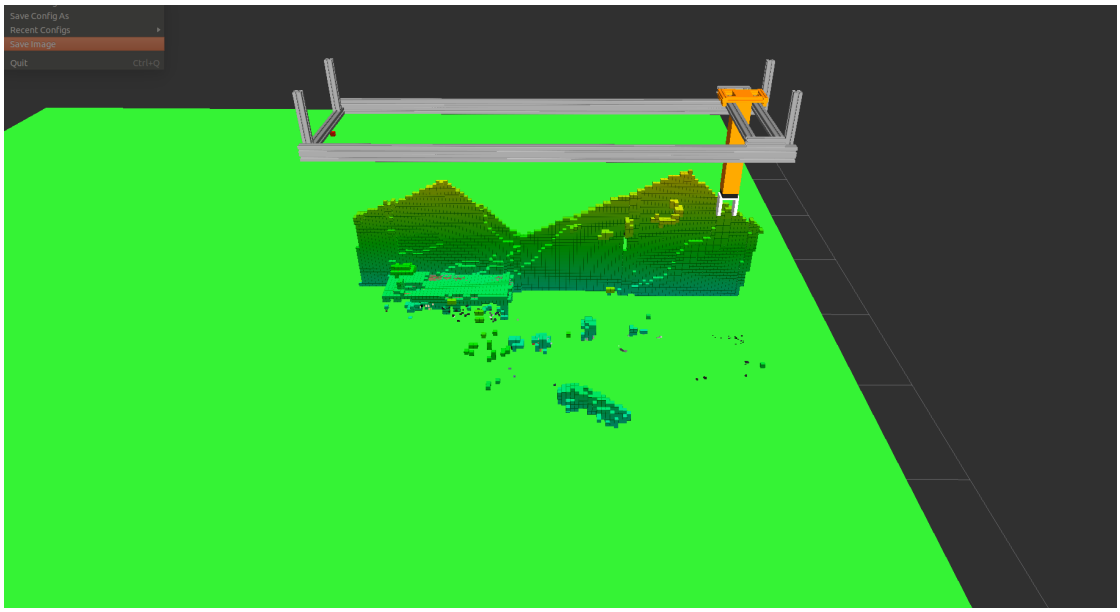

Figure 12: Reconstruction after filtering the table. The box collision object filters points from the table plane to the floor.

6.5.1 Data acquisition

The first step of the plugin consist in joining a point cloud of every camera into a single point cloud. Before joining the point clouds, they are processed by a radius filter that will discard points that don't have at least 4 neighbours in a radius of 2.5cm. The joining of point clouds gives a point cloud of all the visible scene, and therefore a point cloud where the table is complete. This operation is repeated N times (where N is a parameter to be determined by the user) to obtain N point clouds of the complete visible scene.

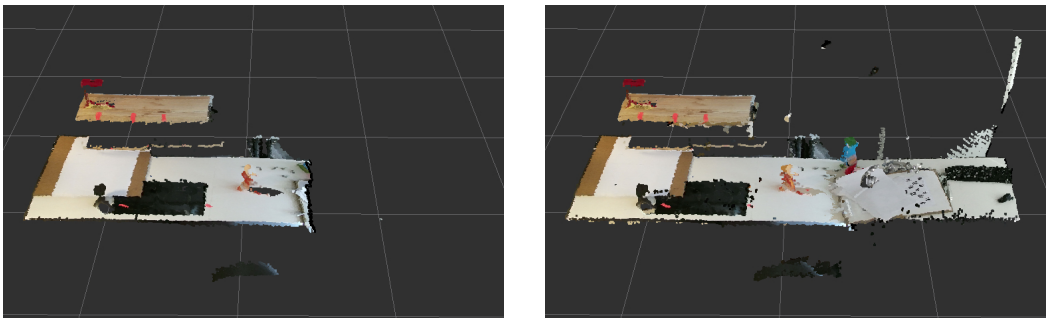

Figure 13: Point cloud of one camera and the final point clouds together.

6.5.2 Detecting the table plane

To every point cloud obtained, the PROSAC method [3] will be used to search for a perpendicular plane to the z axis.

The PROSAC method consist in obtaining a set of tentative correspondences using the linear ordering obtained from a real-valued similarity function and then the samples are semi-randomly drawn from progressively larger sets of the tentative correspondences. The process finish when a stopping criterion is reached.

When the table is found in each point cloud, the average of the coefficients of the table planes is calculated to be able to create the collision object. The PROSAC implementation used is the one provided by the point cloud library.

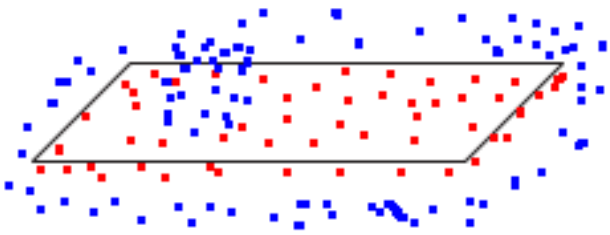

Figure 14: Point fitting to a planar model. Red points are inliers.

6.6 Temporal filter

Even that the filtering of points performed reduces the computation time needed per frame, the raycast technique continues to be too expensive and it's not possible to achieve real time with the point cloud extracted from the kinect 2 cameras at the desired octomap resolution.

As stated before, the planing used in the robots treats the unknown areas as free areas. It's possible to take advantage of this and the probabilistic approach of octomap.

The scene will be modelled with the assumption that the scene tends to be free. This means that for a voxel to be marked as occupied it has to have a point in the point cloud that lie in it a certain amount of frames. If an occupied voxel don't receive a point that lie in it another certain number of frames it will be marked as free.

The first thing to be aware is that the octomap don't work with direct probabilities. Instead it uses the following formula.

$$\log\left[\frac{P(n)}{1-P(n)}\right]$$

having that a percentage less than 0.5 is negative and a percentage greater is positive.

Having this in mind, the technique consist in that every time that a point cloud is received, the corresponding voxels to the points increase their probability to be occupied by an ϵ percentage. When all the point cloud is processed, the probability of all the voxels in the octomap to be occupied is lowered by the temporal filter by an α percentage, where $\alpha < \epsilon$. The points with positive value will be considered occupied, while the values with negative value will be considered free.

This technique makes that a certain minimum and maximum limit values must be defined in order to don't make virtually impossible to mark a occupied voxel as

free if it has been occupied for a large time and the inverse case, a long time free voxel has to be able to be marked as occupied in a prudent time.

Depending on this values, α can be selected in a way that an occupied point could be marked as free only if it has stopped to appear N times. The same can be done for ϵ, that can be selected to make that a free point pass to be marked occupied only if it has been occupied M times.

$$min < 0$$
$$N = ceil\left(\frac{max}{\alpha}\right)$$
$$M = ceil\left(\frac{abs(min)}{(\epsilon - \alpha)}\right)$$

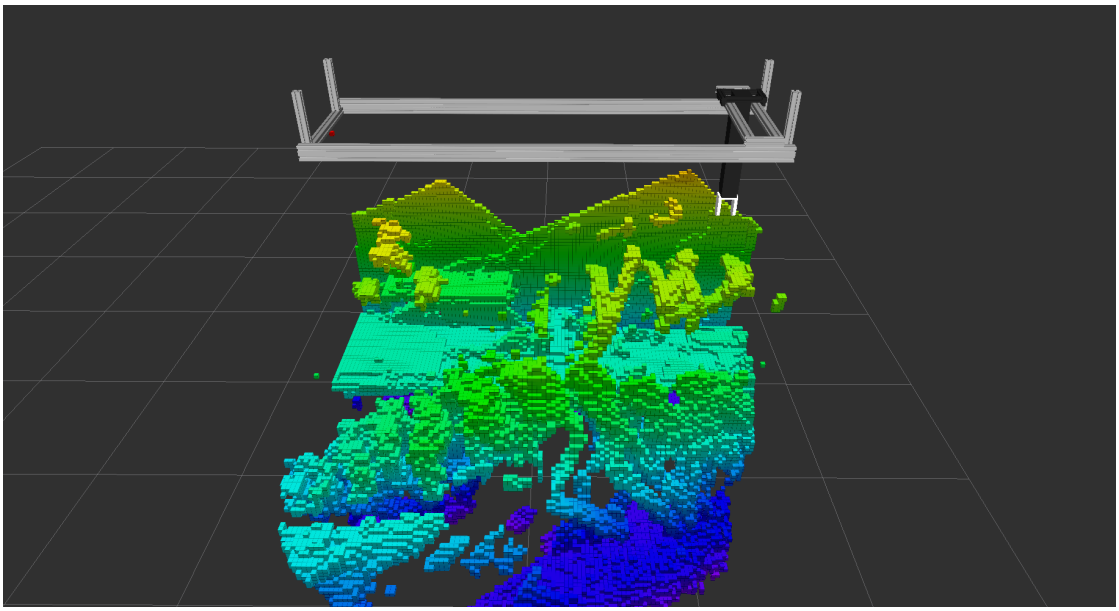

Figure 15: Reconstruction using the temporal filter with high M when a person walks from a side of the table to the other side with the arm extended over the table. The points fail to be cleaned in a reasonable time.

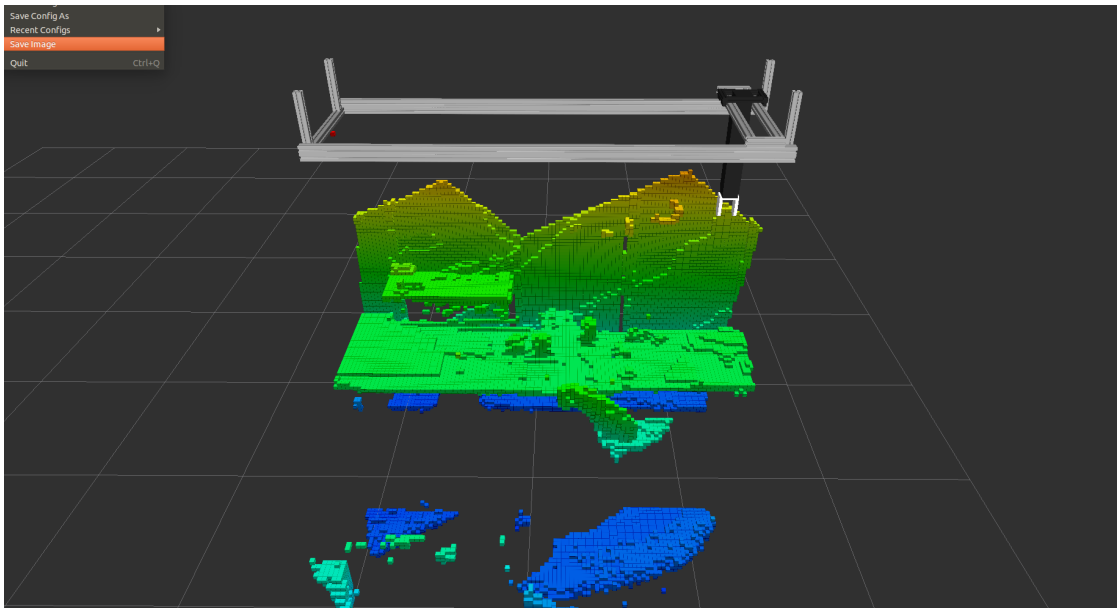

Figure 16: Reconstruction using the temporal filter with N=7 and M=3.

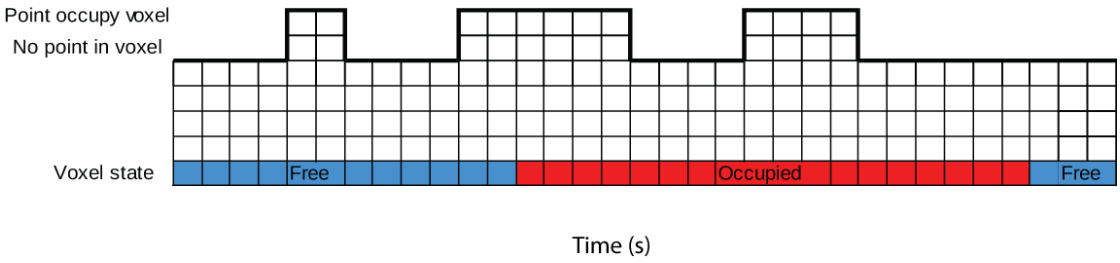

Figure 17: State of a voxel depending if it is marked as occupied in a point cloud or not wit parameters set in a way that it is satisfied that N=7, M= 3.

6.7 Sparse point filter

To be able to delete some "floating" points that sometimes appear due bad depth measurements in the point cloud, a filter has been developed. This filter can be activated in two modes. The normal mode will filter new occupied voxel checking

for how many neighbours it has in a 6-connected 3D space in the current octomap. The accurate mode will search for neighbours also in the new data set being added.

As it can be a computational expensive, this filter can be activated or deactivated by the user on the cameras. In this way it is possible to lower the computational load if the filtering is not necessary.

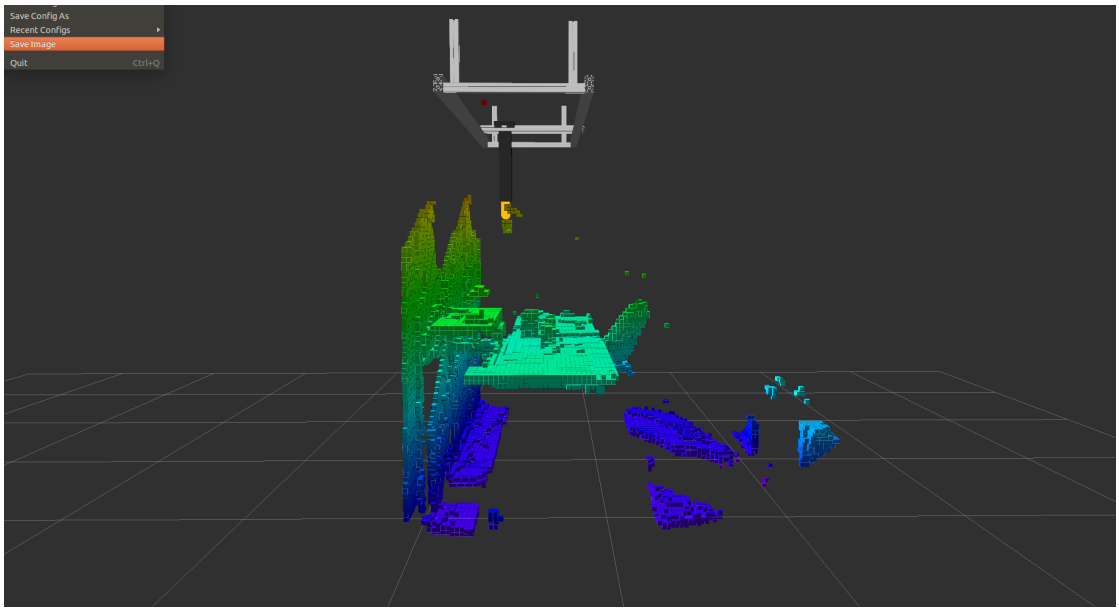

Figure 18: Sparse point deactivated. Some occupied voxels are floating above the table.

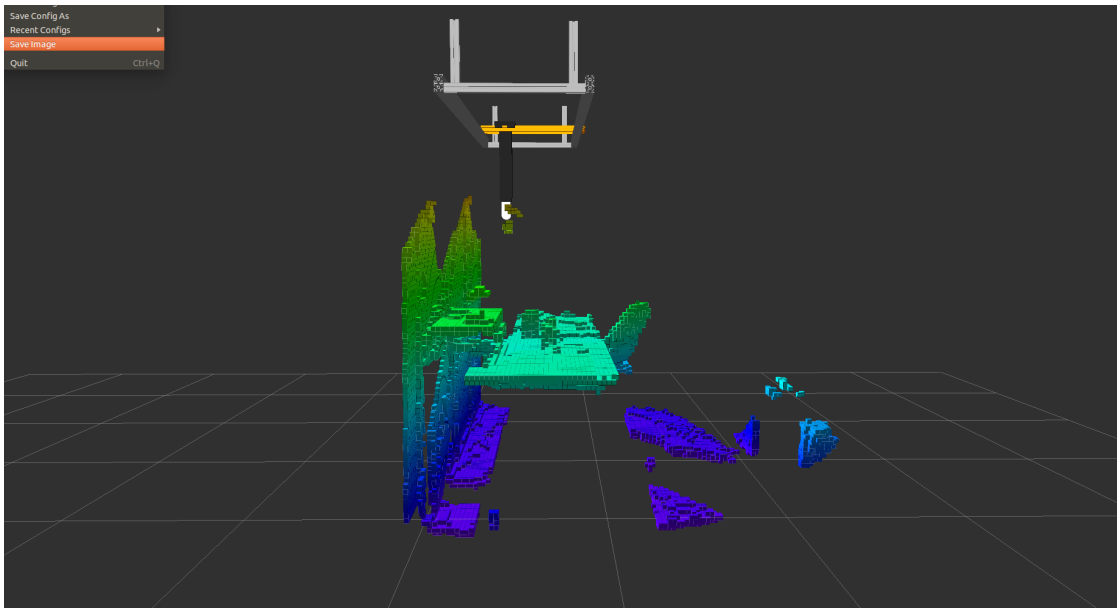

Figure 19: Sparse point activated. The occupied floating voxels are gone.

6.8 Remote perception

Even that the architecture that it is used by ROS MoveIt! is designed for a centralized environment where a single machine performs the perception, the robot planning, etc, the reconstruction plugin has been developed in a way that it is possible that the octomap can be generated in a computer dedicated to this task and then distribute the octomap to another computer running ROS MoveIt! controlling a robot.

This is done by giving the option to publish a message with the octomap in a topic where the other computer is subscribed.

This allows to move the computing intensive task of the perception to a computer with more resources that the one that controls the robot. The counterpart is that the possible network lag can produce the lost of the real time performance. The user has to decide which option is best for his case, whether to use the plugin

in a single machine or distribute the perception load in several computers.

6.9 Acceleration using GPU with openCL 1.1

An attempt to accelerate the reconstruction using the implementation of the temporal filter technique is done.

The profiling tool callgrind is used to determine which parts of the code are the most computationally expensive and try to compute it in the GPU. This part corresponds to the validation of the points (check if the point 3D components are not NaN) and the operation of transforming the points from the camera reference frame to the world reference frame.

Unfortunately, the overhead associated with the copy of the point cloud to the GPU is greater than the time to process the point cloud without using the GPU, and this option had to be discarded.

7 Results

7.1 Reconstruction accuracy

It is desired to know if the reconstruction is accurate to the reality. There are two ways to measure how accurate is the reconstruction: the position accuracy of the objects and it's size accuracy. The first will measure the absolute error of the reconstruction and the second the relative error.

To measure the reconstruction error of the position, the position of a table corner is measured and compared with the position given in the world reference frame of the reconstruction. For the reconstruction error of the size, the table is measured and compared with the size given by the reconstruction.

All the data gathered from the reconstruction is obtained using rviz with the

tools measure and publish point.

7.1.1 Position error

A table corner is located at the position $X = 1.01m \quad Y = 0.73m \quad Z = 0.8m$ of the world reference frame. The same corner in the reconstruction is at $X = 1.049m \quad Y = 0.776m \quad Z = 0.82m$ of the world reference frame, giving an absolute error of 0.039m in X, 0.046m in Y and 0.02 in Z. The relative error is 0,038613861 in X, 0,063013699 in Y and 0,025 in Z.

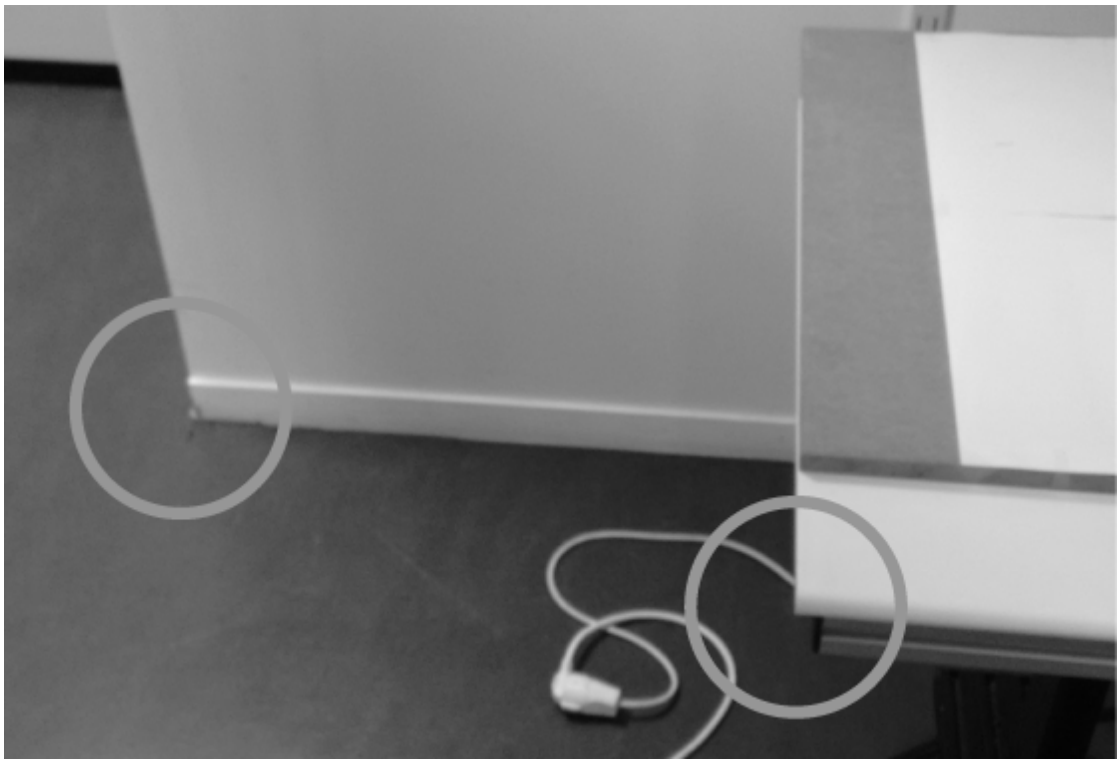

Figure 20: World reference frame in the floor, and the table corner used to measure the absolute error.

7.1.2 Size error

The table measured size is $X = 0.61m \quad Y = 2.635m \quad Z = 0.015m$. The table size in the reconstruction is $X = 0.621m \quad Y = 2.64656m \quad Z = 0.075m$, giving an absolute error of 0.011 in X, 0,01156 in Y and 0.06 in Z. The relative error is 0,0180327869 in X, 0,004387097 in Y and 4 in Z.

7.2 Reconstruction speed

The reconstruction speed of several configurations of the developed techniques will be checked.

To test the reconstruction speed, it is measured the time that it is necessary to process 1000 frames and then the time needed will be divided by 1000 to obtain the FPS (frames per second) that the computer is able to compute.

To obtain the measurements, after starting the system, the firsts 600 frames will not be taken in account. This is is done in order to not have problems of cache misses or pagination problems that are present when the algorithm start and to don't have the table plugin working in the plane detection, using compute resources, while the test is being performed.

After the processing the 600 frames the current time is captured and after processing 1000 frames the time is captured again to be able to calculate the elapsed time and obtain the FPS.

In all the test the remote stream of the octomap is deactivated. The camera max range for camera 1 is 2.2m and for the camera 2 is 3.5m. The octomap resolution is 2.5cm, near 1 inch. This is a compromise between a resolution of 5cm that in the test had a good speed but large errors and 1 cm, that performs an accurate reconstruction at the cost of speed.

The computer used is composed of two Intel Xeon Workstation T7500 with 32 GB RAM and a Nvidia Tesla C2050.

7.2.1 Raycast technique

Two main configurations will be used to compare the results of using the table filter and not using them. Also, a test will be done to have an approximate idea of which is the maximum speed that the technique can achieve if it only process one camera.

—First configuration—

- Octomap resolution of 0.025m
- Table filtering not used.
- Sparse point filter deactivated.

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	4.303373122	3.9840436185	4.1437083702
Camera 2	2.2484480352	2.2369989224	2.2427234788

Table 2: Point subsample = 1 in the two cameras

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	4.0315911776	5.1401731911	4.5858821844
Camera 2	2.9278139535	2.4285868747	2.6782004141

Table 3: Point subsample = 2 in the two cameras

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	5.6617671941	5.6733024259	5.66753481
Camera 2	3.4936209538	3.1829920259	3.3383064899

Table 4: Point subsample = 4 in the two cameras

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	7.0117687701	7.147392674	7.0795807221
Camera 2	3.4966204744	3.1290306119	3.3128255431

Table 5: Point subsample = 8 in the camera 1 and point subsample = 4 in camera 2

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	8.26223	7.828456	8.0452

Table 6: Test of camera 1 alone. Point subsample = 8.

—**Second configuration**—

- Octomap resolution of 0.025m
- Table filtering activated.
- Sparse point filter deactivated.

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	4.8305602496	5.1985862819	5.0145732657
Camera 2	2.7132146127	2.6279128637	2.6705637382

Table 7: Point subsample = 1 in the two cameras

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	5.4460202347	4.3190361059	4.8825281703
Camera 2	2.6787025343	2.7840410736	2.731371804

Table 8: Point subsample = 2 in the two cameras

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	4.8952115354	4.9336403663	4.9144259508
Camera 2	3.7405941917	4.395550567	4.0680723794

Table 9: Point subsample = 4 in the two cameras

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	11.4451345657	10.4323015352	10.9387180505
Camera 2	4.4557164367	5.0331396004	4.7444280185

Table 10: Point subsample = 8 in the camera 1 and point subsample = 4 in camera 2

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	9.111174	9.771894	9.441534

Table 11: Test of camera 1 alone. Point subsample = 8.

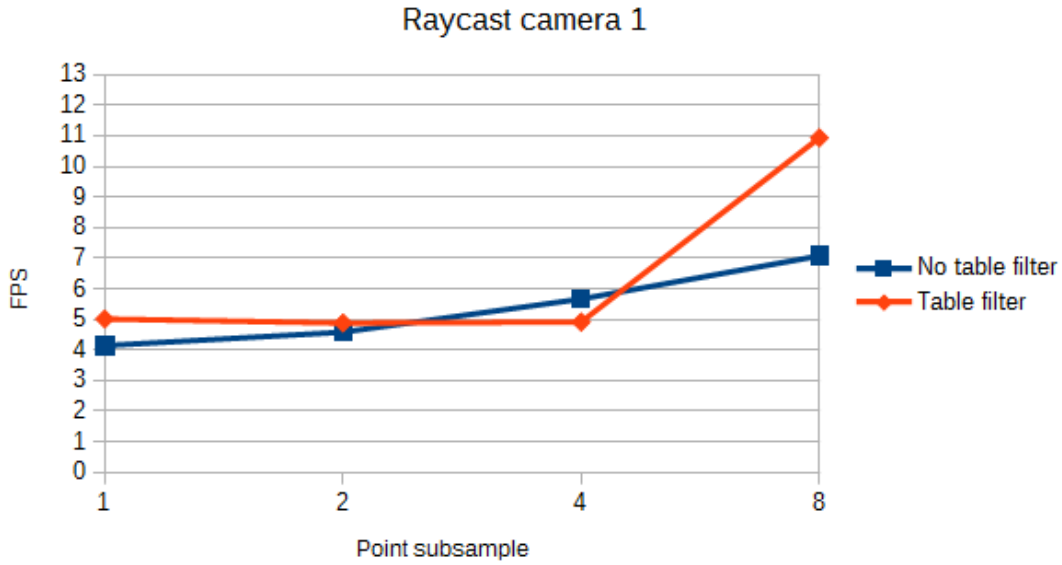

Figure 21: Raycast average FPS of camera 1 for both configurations

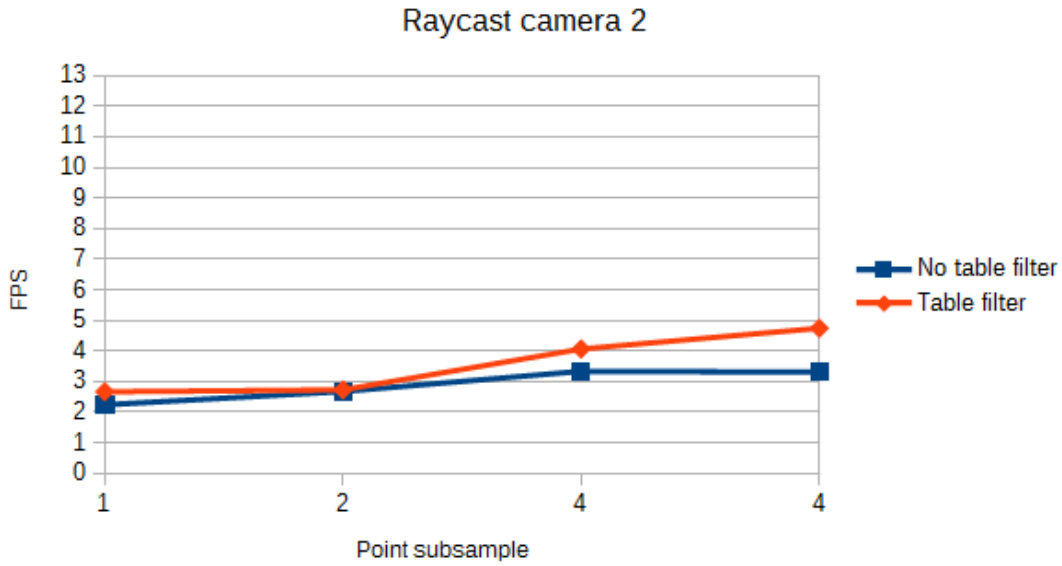

Figure 22: Raycast average FPS of camera 2 for both configurations

7.2.2 Temporal filter technique

The parameters of the temporal filter has been set to mark a free voxel as occupied if the voxel is occupied for three frames. An occupied voxel will be set free after seven frames of being free. This is to act fast to new obstacles while rejecting bad depth measurements and to be sure that an obstacle it's no longer in a voxel.

Three groups of two configurations will be presented. The first group is used to see the effect of the table filter in the temporal filter reconstruction and have an approximated idea of the maximum speed that can be achieved with this technique if it is used with a single camera. The second will test the sparse filter in normal mode while the third will test the sparse filter in accurate mode.

—First configuration—

- Octomap resolution of 0.025m
- Table filtering not used.
- Sparse point filter deactivated.

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	8.30108	8.48014	8.39061
Camera 2	7.7693	7.9386	7.85395

Table 12: Point subsample = 1 in the two cameras

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	10.1117	9.8606	9.98615
Camera 2	8.92999	9.15895	9.04447

Table 13: Point subsample = 2 in the two cameras

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	10.9489	10.9535	10.9512
Camera 2	10.3799	10.3717	10.3758

Table 14: Point subsample = 4 in the two cameras

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	11.5976	11.6266	11.6121
Camera 2	10.5655	10.4869	10.5262

Table 15: Point subsample = 8 in the camera 1 and point subsample = 4 in camera 2

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	18.4722	19.0635	18.76785

Table 16: Test of camera 1 alone. Point subsample = 8

—**Second configuration**—

- Octomap resolution of 0.025m
- Table filtering activated.
- Sparse point filter deactivated.

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	8.52496	9.66713	9.096045
Camera 2	7.75353	9.08145	8.41749

Table 17: Point subsample = 1 in the two cameras

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	10.707	10.6728	10.6899
Camera 2	10.1644	10.1646	10.1645

Table 18: Point subsample = 2 in the two cameras

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	11.7103	11.589	11.64965
Camera 2	11.1182	11.2688	11.1935

Table 19: Point subsample = 4 in the two cameras

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	12.0928	12.1436	12.1182
Camera 2	11.4226	11.4206	11.4216

Table 20: Point subsample = 8 in the camera 1 and point subsample = 4 in camera 2

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	18.5013	18.2326	18.36695

Table 21: Test of camera 1 alone. Point subsample = 8

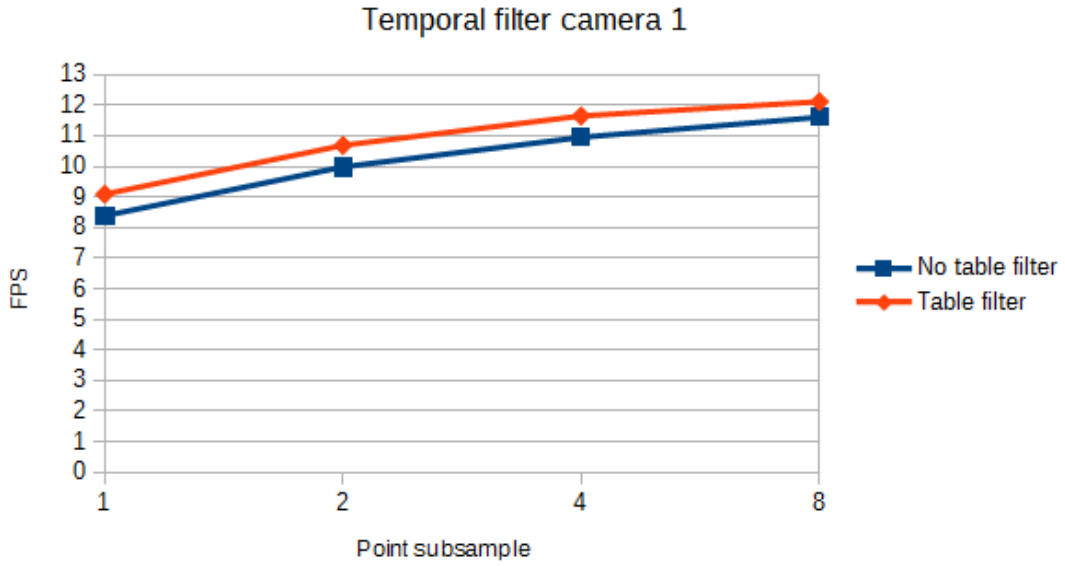

Figure 23: Temp. filter average FPS of camera 1 for the 1st and 2nd configurations

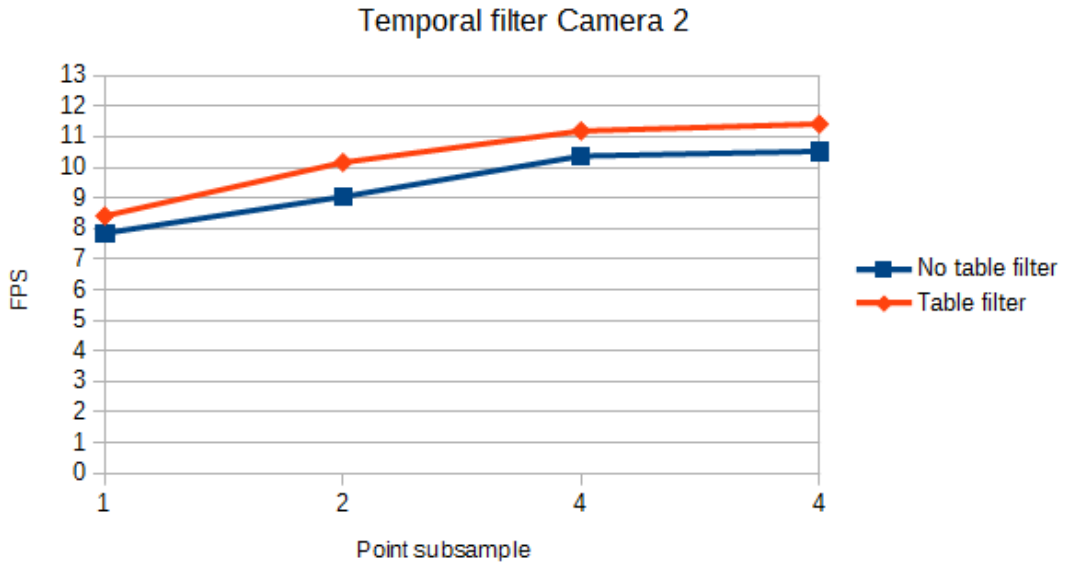

Figure 24: Temp. filter average FPS of camera 2 for the 1st and 2nd configurations

—Third configuration—

- Octomap resolution of 0.025m
- Table filtering activated.
- Sparse filtering activated in normal mode.
- Sparse filtering done at every frame.

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	9.41725	9.44772	9.432485
Camera 2	8.86417	8.92946	8.896815

Table 22: Point subsample = 1 in the two cameras

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	10.582	10.8222	10.7021
Camera 2	10.1935	10.2166	10.20505

Table 23: Point subsample = 2 in the two cameras

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	11.3773	11.2969	11.3371
Camera 2	10.9382	10.9474	10.9428

Table 24: Point subsample = 4 in the two cameras

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	11.7952	12.0939	11.94455
Camera 2	11.0466	11.2488	11.1477

Table 25: Point subsample = 8 in the camera 1 and point subsample = 4 in camera 2

—Fourth configuration—

- Octomap resolution of 0.025m
- Table filtering activated.
- Sparse filtering activated in normal mode.
- Sparse filtering done every 30 frames.

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	9.34888	9.63653	9.492705
Camera 2	8.79939	8.93974	8.869565

Table 26: Point subsample = 1 in the two cameras

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	10.833	10.7155	10.77425
Camera 2	10.3406	10.2459	10.29325

Table 27: Point subsample = 2 in the two cameras

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	11.539	11.4944	11.5167
Camera 2	11.1387	10.889	11.01385

Table 28: Point subsample = 4 in the two cameras

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	12.0481	12.0621	12.0551
Camera 2	11.5542	11.4601	11.50715

Table 29: Point subsample = 8 in the camera 1 and point subsample = 4 in camera 2

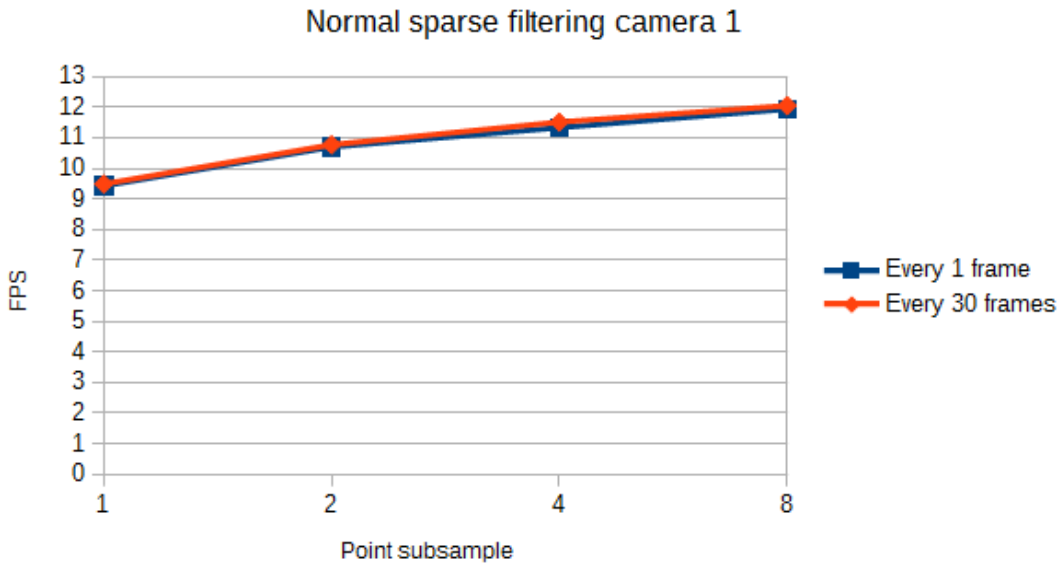

Figure 25: Temp. filter average FPS of camera 1 for the 3rd and 4th configurations

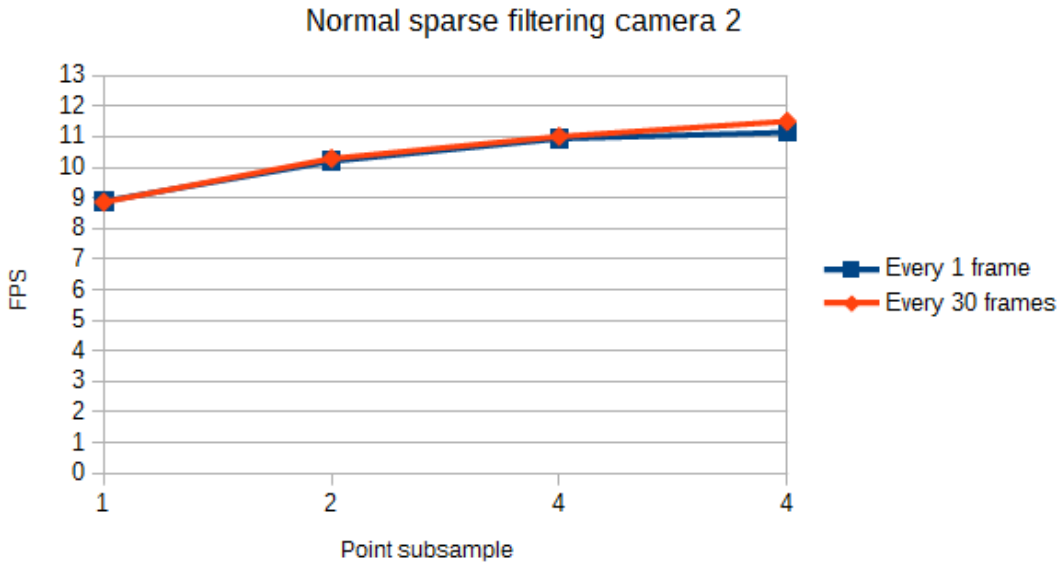

Figure 26: Temp. filter average FPS of camera 2 for the 3rd and 4th configurations

—Fifth configuration—

- Octomap resolution of 0.025m
- Table filtering activated.
- Sparse filtering activated in accurate mode.
- Sparse filtering done at every frame.

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	8.9837	9.19904	9.09137
Camera 2	8.5487	8.77733	8.663015

Table 30: Point subsample = 1 in the two cameras

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	10.5547	10.5058	10.53025
Camera 2	9.84604	9.98785	9.916945

Table 31: Point subsample = 2 in the two cameras

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	10.9131	10.7751	10.8441
Camera 2	10.4314	10.5309	10.48115

Table 32: Point subsample = 4 in the two cameras

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	11.6132	11.8926	11.7529
Camera 2	10.7931	10.9637	10.8784

Table 33: Point subsample = 8 in the camera 1 and point subsample = 4 in camera 2

—Sixth configuration—

- Octomap resolution of 0.025m
- Table filtering activated.
- Sparse filtering activated in accurate mode.
- Sparse filtering done every 30 frames.

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	9.43434	9.47405	9.454195
Camera 2	8.80398	8.83626	8.82012

Table 34: Point subsample = 1 in the two cameras

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	10.6386	10.6375	10.63805
Camera 2	10.2044	10.1564	10.1804

Table 35: Point subsample = 2 in the two cameras

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	11.5227	11.5424	11.53255
Camera 2	11.3181	11.08	11.19905

Table 36: Point subsample = 4 in the two cameras

Camera	FPS 1st test	FPS 2nd test	FPS average
Camera 1	11.9647	12.0733	12.019
Camera 2	11.2227	11.3368	11.27975

Table 37: Point subsample = 8 in the camera 1 and point subsample = 4 in camera 2

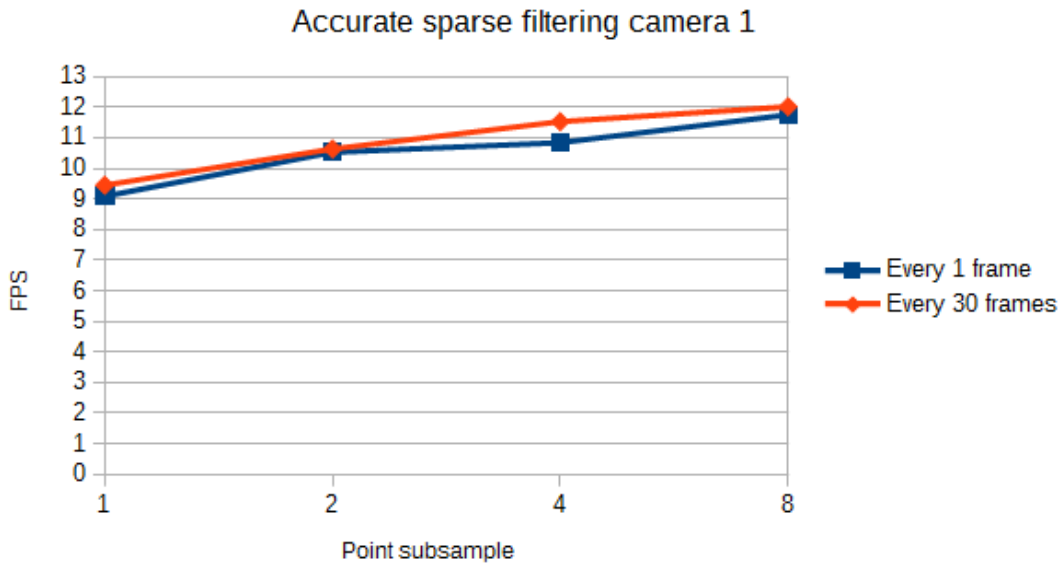

Figure 27: Temp. filter average FPS of camera 1 for the 5th and 6th configurations

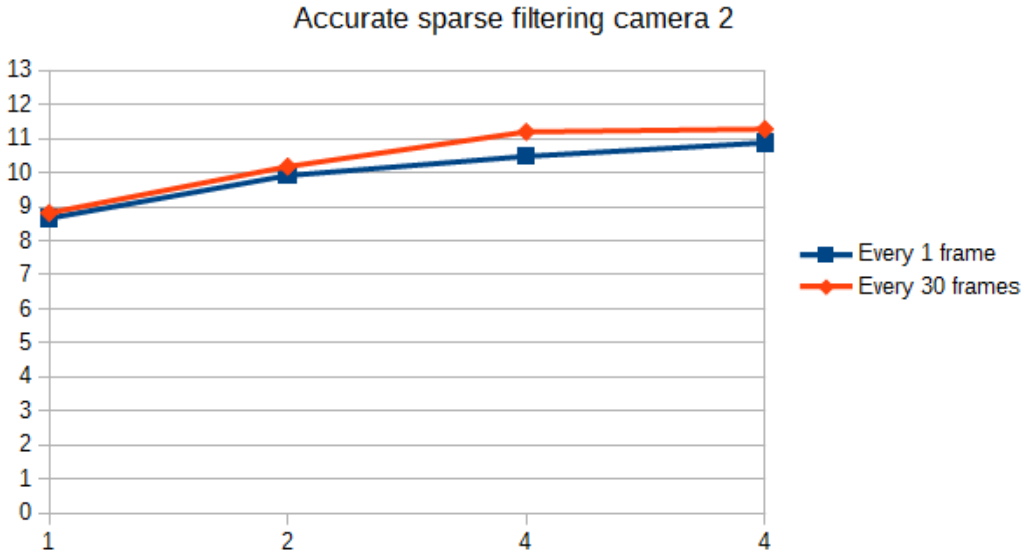

Figure 28: Temp. filter average FPS of camera 2 for the 5th and 6th configurations

7.2.3 Techniques comparison

The next plot has the sum of the FPS of the cameras for the corresponding technique to asses their performance and how they are affected by the point subsample. The data is gathered from the results with the table filter active.

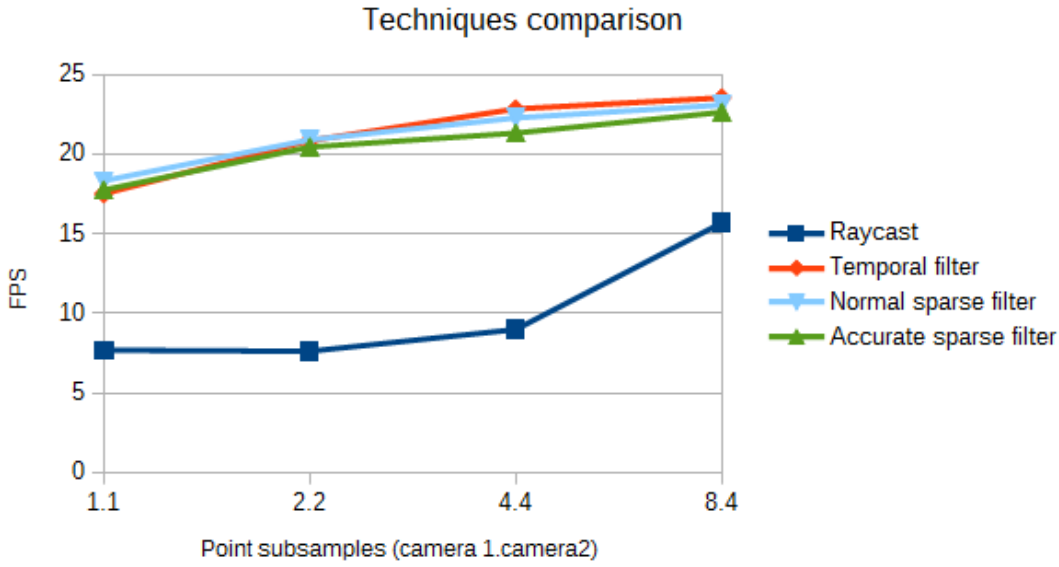

Figure 29: Speed comparison of the different techniques.

7.3 Table detection results

7.3.1 Table detection error

It is desired to calculate the error of the table detection plugin.

The table has a little skew, but the plane at which it lies must be close to the plane (0, 0, 1, 0.80). This plane coefficients will be compared against the given output to have an approximated value of the table detection error.

The plane coefficients detected by the table detection plugin are(0.00407097, 0.00344974, 0.999982, 0.821944).

The approximate error in the plane detection is 2.1944cm, while the relative error is 0,026697683.

7.3.2 Table detection speed

The plugin needs about 1.6 seconds to process every point cloud. Perform the average is almost instantaneous.

It is possible to calculate the approximate time that the plugin need to detect the table with the following formula

$$Detection\_time = 1.6 * N * Number\_of\_cameras$$

where N is the number of complete point cloud scenes to collect as commented in section 6.5.1.

7.4 Valid points results

It is desired to know why camera 1 always process the frames faster than camera 2.

As the profiling tool callgrind marks that the critical parts of the code (the parts where most of the time is spend) of the reconstruction is the validation of the points and their transform to the world reference frame, a test to determine how many valid points each camera obtain is performed. This will give approximately how many point transformations are performed in each camera.

The test consist in counting how many valid points exist in every frame and output the average valid points for all the processed frames.

Camera	Total points in cloud	Average valid points in cloud
Camera 1	518400	311271.4
Camera 2	518400	361487.2

Table 38: Point subsample = 8 in the camera 1 and point subsample = 4 in camera 2
Giving a difference of 50215,8 points to process.

8 Conclusions

8.1 Reconstruction accuracy

From the results can be seen that the maximum position error is 0.046m in Y. Also that the maximum size error is 0.06 in Z. As the world is discretized in voxels, the objects in the reconstruction are larger. This is a good thing in the view of robot planning, as the robot will have a virtual safe distance provided directly by the reconstruction being a little bit bigger. Nevertheless, the position error makes necessary to place at least 0.05m more of this safe distance to avoid possible collisions with misplaced objects. The relative error in size in Z (4) is expected. The size to be measured in Z is smaller than the size of the voxels. This will always lead to have a big amount of error in those cases. Moreover, depending in the position where the table lie, the thickness of the table can lie in two rows of voxels instead of one containing it completely, making the error even bigger.

8.2 Reconstruction speed

For our porpoises, the obtained speed of reconstruction using the temporal filtering technique is enough to name it real-time.

There is a robot that operates at a maximum velocity of 0.2m/s. Configuring

the temporal filter parameters to mark a free area as occupied if three frames mark a voxel occupied the lowest FPS achieved, 7.7, would reconstruct a new obstacle in 0.389s. In this time the robot could only move 7.7cm, which is a very short distance. Moreover, if it is necessary it is possible to isolate a camera to a computer and use the remote capability to obtain better FPS as it has been checked in table 16 and table 21.

The raycast technique should only be used in cases that an accurate model of the unknown areas must be obtained or where the scene permits using the technique with better FPS than the obtained in this work. For example scenes where it is possible to have a high value of point subsample set.

Finally, in section 7.4 it is discovered that the the reconstruction speed is affected by how many valid points exist.

8.3 Table plane detection

The table plane detection helps to improve the performance of the reconstruction, and it's even possible to detect the table while there are objects placed over it.

Unfortunately, this technique is high compute expensive and it can take up to 26 seconds to create the collision object if it has to average 10 complete point clouds. This makes that the system can not take advantage of the speed up provided by the plugin after a certain start-up time.

This normally will not be a problem, but it has to be taken into account.

9 Future work

9.1 Camera frame to octomap LUT (Look up table)

As the current major compute time is in transforming the point from the camera reference frame to the world reference frame to know where in which voxel in the octomap the point lies, but this is not able to be done by the GPU, a possible solution to speed up this part could be performing a LUT that could map a 3D point in the camera reference frame to the corresponding octomap voxel without having to actually perform the transformation.

9.2 GPU raycast

An attempt to perform the raycast using the GPU could be a good way to achieve a good model in the case that treating the unknown space as free space is unacceptable.

9.3 Octomap share between robots

In case that there are various robots controlled by ROS MoveIt!, it could be a good idea to save resources that the octomap generated by one robot could be send to the other robots filtering the robot at which the octomap is sent and put as obstacle the robot that generated it.

9.4 Object shape recognition

It could be possible to filter the voxels over the table and try to locate the objects over it. With this the robot could use the pick and place function to manipulate objects.

References

[1] Point cloud library. `http://pointclouds.org/`.

[2] D.A. Butler, S. Izadi, O. Hilliges, D. Molyneaux, S. Hodges, and D.Kim. Shake'n'sense: reducing interference for overlapping structured light depth cameras. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, page 1933–1936. ACM, 2012. http://research.microsoft.com/apps/pubs/default.aspx?id=171706.

[3] Ondrej Chum and Jiri Matas. Matching with prosac " progressive sample consensus. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, pages 220–226, Washington, DC, USA, 2005. IEEE Computer Society.

[4] Florian Faion, Simon Friedberger, Antonio Zea, and Uwe D. Hanebeck. Intelligent sensor-scheduling for multi-kinect-tracking. In *IROS*, pages 3993–3999. IEEE, 2012.

[5] J. Geng. Structured-light 3d surface imaging: a tutorial. *Advances in Optics and Photonics*, (3):128–160, 2011.

[6] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. Software available at `http://octomap.github.com`.

[7] Texas instruments. Time-of-flight cameras. `www.ti.com/lit/wp/sloa190b/sloa190b.pdf`.

[8] Kurt Konolige and Patrick Mihelich. Technical description of kinect calibration. `http://wiki.ros.org/kinect_calibration/technical`. Wiki last edition date: 12-27-2012 at 08:02:31.

[9] Wim Lemkens, Prabhjot Kaur, Koen Buys, Peter Slaets, Tinne Tuytelaars, and Joris De Schutter. Multi rgb-d camera setup for generating large 3d point clouds. In *IROS'13*, pages 1092–1099, 2013.

[10] Scott Niekum. Ros package ar_track_alvar. `http://wiki.ros.org/ar_track_alvar`.

[11] Pedro Santana. Ros package ros-tf-graph. `https://github.com/phrqas/ros-tf-graph`.

[12] Yannic Schröder, Alexander Scholz, Kai Berger, Kai Ruhl, Stefan Guthe, and Marcus Magnor. Multiple kinect studies. Technical Report 09-15, ICG, October 2011.

[13] Thiemo Wiedemeyer. Ros package iai_kinect2. `https://github.com/code-iai/iai_kinect2`.